# ICS 第四次实验实验报告

李浩然　2025 年 11 月 26 日

## 1　实验目的

本实验旨在实现一个基于 LC-3 汇编语言的递归程序，计算目标坐标 $(N, M)$ 的配送推荐值。具体目标包括：

1. 基于给定公式计算到达 $(N, M)$ 的最小步数 $Steps(N, M) = N + M$；

2. 基于递归公式计算到达 $(N, M)$ 的唯一路径数 $Routes(N, M)$（边界条件：$i = 0$ 或 $j = 0$ 时 $Routes = 1$，否则 $Routes(i, j) = Routes(i - 1, j) + Routes(i, j - 1)$）；

3. 按规则计算推荐值（推荐值 $= 5 \times Routes(N, M) - Steps(N, M)$），并将结果存储到内存地址 x3200；

4. 验证递归程序的正确性，分析递归与迭代方法的效率差异。

## 2　实验过程

### 2.1　算法设计

- 递归函数 $PATH$：接收参数 $X(R1)$、$Y(R2)$，返回路径数 $Routes(X, Y)$ 至 $R0$；
- 边界处理：当 $X = 0$ 或 $Y = 0$ 时，直接返回 1；
- 递归调用：$Routes(X, Y) = PATH(X - 1, Y) + PATH(X, Y - 1)$；
- 推荐值计算：$= 5 \times Routes - Steps$（$Steps = X + Y$）；

**栈操作设计**：栈起始地址 x4000，递归时保存 $R1(X)$、$R2(Y)$、$R3()$、$R7()$，返回时恢复；

## 3　实验结果

通过 LC-3 评测机评测，源代码如下：

```
.ORIG x3000
LD R0, NUM0;R0=0
LD R3, X
LDR R1, R3, #0
LD R3, Y
LDR R2, R3, #0
;R1 = X, R2 = Y
LD R3, NUM0
LD R4, NUM0
LD R5, ANS_LOC
LD R6, STACK
;R6 = &S[0]
LD R7, NUM0
```

```
JSR PATH

ADD R4, R1, R2
;S(i,j)
ADD R3, R0, R0
ADD R3, R3, R0
ADD R3, R3, R0
ADD R3, R3, R0
;5*R(i,j)
NOT R4, R4
ADD R4, R4, #1
ADD R3, R3, R4
STR R3, R5, #0
;得到答案储存后跳转到HALT
BRnzp STOP

PATH

ADD R6, R6, #-4
STR R1, R6, #0
STR R2, R6, #1
STR R3, R6, #2
STR R7, R6, #3
;存这几个数

ADD R3, R1, #0
BRz IF0
ADD R3, R2, #0
BRz IF0
;X,Y是0就跳边界条件
LD R3, NUM0

ADD R1, R1, #-1
;PATH(X-1, Y)
JSR PATH

ADD R3, R0, #0
ADD R1, R1, #1
ADD R2, R2, #-1
;PATH(X, Y-1)
JSR PATH
```

```
;PATH(X,Y) = PATH(X-1,Y) + PATH(X,Y-1)
ADD R0, R0, R3
BRnzp BACK

IF0
LD R0, NUM1
;R(i,j)=1
BACK

LDR R1, R6, #0
LDR R2, R6, #1
LDR R3, R6, #2
LDR R7, R6, #3
ADD R6, R6, #4
;复原这几个数

RET

STOP

HALT
NUM0    .FILL #0
NUM1    .FILL #1
X   .FILL x3100
Y   .FILL x3101
ANS_LOC     .FILL x3200
STACK   .FILL x4000

.END
```

# 4 讨论

## 4.1 迭代方法比基础递归方法更高效的原因

迭代方法优于基础递归方法的核心原因包括：

1. **避免重复计算**：递归实现中，相同子问题（如计算 $PATH(3,2)$ 和 $PATH(2,3)$ 时均会调用 $PATH(2,1)$）会被多次重复求解，时间复杂度为指数级 $O(2^{N+M})$；而迭代方法按顺序计算每个子问题，每个子问题仅求解一次，时间复杂度降至多项式级 $O(N \times M)$。
2. **消除栈操作开销**：递归依赖调用栈保存寄存器状态（如 $R1$、$R2$、$R7$）和返回地址，每次递归调用/返回均需执行栈入栈出操作，增加时间与内存开销；迭代通过循环和显式状态管理，无需栈操作，开销显著降低。

3. **无栈溢出风险**：递归深度受栈空间限制（尤其当 $N$ 或 $M$ 较大时），易触发栈溢出；迭代仅使用固定内存（如动态规划的数组），完全规避该风险。

## 4.2 程序效率优化方案

基于递归的局限性，可通过以下方式提升程序效率：

1. **动态规划(迭代方式)**:使用二维数组(或一维数组优化空间)存储子问题结果 $Routes(i,j)$，从边界条件（$i=0$ 或 $j=0$）开始，逐步迭代计算至目标坐标 $(N,M)$，确保每个子问题仅求解一次。

2. **记忆化递归**：添加缓存（如内存表）存储已计算的 $PATH(i,j)$ 结果，递归前先查询缓存，若结果已存在则直接复用，在保留递归逻辑的同时将时间复杂度降至 $O(N\times M)$。

3. **组合数学公式**：利用组合数原理，$Routes(N,M)$ 等价于从 $N+M$ 步中选择 $N$ 步向右的组合数，即 $Routes(N,M)=\binom{N+M}{N}$，通过迭代乘法直接计算组合数，无需递归或复杂迭代。

4. **尾递归优化**：将递归函数改写为尾递归形式（递归调用为最后一步操作），使编译器/解释器可复用当前栈帧，减少栈空间占用与操作开销。